THE UNIVERSITY OF
MELBOURNE

Department of
Computer Science and Software Engineering

Software Project Management Plan

for

Managing the analysis of massive DNA sequence data

Version: 2.6
October 30, 2009

This document aims to act as a guideline for each team member to follow throughout the entire development life cycle. Members are expected to refer to this document often, so they can all follow the right processes while carrying out their assigned tasks. This document also includes ways to deal with certain situations that may arise during the development life cycle. Parts of this document will be updated continuously throughout the year as new procedures are adapted, old procedures are modified, and new risks to the project are identified and encountered.

Copyright notice

Copyright © 2009, Team D. Permission is granted to reproduce this document for internal Team D use only.

Department of Computer Science and Software Engineering
The University of Melbourne
Victoria
AUSTRALIA
3010

ICT Building
111 Barry Street
Carlton

Tel: +613 8344 1300
Fax: +613 9348 1184
https://www.csse.unimelb.edu.au/

Version: 2.6
October 30, 2009.
https://www.cs.mu.oz.au/SE-projects/s340gd

Credits

This document was written by: Jack Low (wjlow), Lawrence Bang (lbang), Weng Hoe Ng (wengn), and Jinita Patel (jpatel).

Acknowledgments

The following people provided assistance with or were involved in the development of the document: Nick Chadwick and Jason Li.

CONTENTS

ONE

Introduction

1.1 Project Overview

A GUI that wraps around BLAST is to be created. The GUI should allow the selection and manipulation of input files and reference files to be BLASTed against each other. The system must run in multiple threads to speed up the BLAST process and allow the display of output from threads that have finished running.

1.2 Evolution of the SPMP

This document is subject to change over time throughout the software development. Improvements and alterations will be made to reflect the current processes and plans applied. These involve the following:

- Project Organisation
- Requirements Plan
- Design Plan
- Implementation Plan
- Configuration Management Plan
- Documentation Plan
- Quality Assurance Plan
- Resource Management Plan
- Risk Management Plan

A changelog section will record any changes made which will lead to a SPMP with a new version number. However, if these changes lead to schedule clashing of the project, the team will consult the supervisor before these plans are applied and admitted in the SPMP. This will be done in supervisor meetings.

1.3 Reference Material

http://www.softwaretestinghelp.com/how-to-write-good-bug-report/ - Bug report template

TWO

Project Organisation

2.1 Project Team Organisation

Due to the small size of the team, the team is employing a rather flat structure. A hierachical structure would slow down vertical communication, causing delays in completing simple tasks. A flat structure would mean that it is easier for team members to cooperate with each other. Any problems faced can go straight to the project manager. This way, communication will not be lost like it might in the case of a more hierachical structure.

Team Member	Role(s)	CSSE Login	Contact Details
Jack Low	Project Manager	wjlow	0401271484, jacklow@gmail.com
Lawrence Bang	SAG, Assistant Project Manager, Lead Coder	lbang	0403002530, backlashed@gmail.com
Weng Hoe Ng	Lead Tester	wengn	0423946750, weroth@gmail.com
Jinita Patel	Lead Designer	jpatel	0403662865, patel.jinita@gmail.com

2.1.1 Roles and Responsibilities

This section explains the roles and responsibilities of the team.

2.1.1.1 Project Manager

The project manager is the most important figure in managing the software development team. Delivering a high quality software product to the client depends highly on the project manager.

The responsibilities of a project manager encompass:

- Maintaining a contact link with the supervisor as a representative of the team.
- Gathering suggestions from the team and assign them tasks according to their preference.
- Ensuring adequate records of team decisions, allocated tasks and their predicted and actual completion dates are kept.

- Deciding objectives and priorities at each stage.
- Arranging formal team meetings with supervisor and client.

2.1.1.2 Assistant Project Manager

The assistant project manager helps the project manager in carrying out his tasks. A very important role of the assistant project manager is to monitor the project team and ensure that tasks assigned (whether it be by the project manager or someone else) are being carried out. When the project manager has to be away or is uncontactable for any reason, the assistant project manager will take charge instead.

2.1.1.3 SAG/Technical Officer

The technical officer act as a system administrator and deals with technical issues within the team.

The activities of a technical officer mainly revolves around:

- Assisting team members with technical problems.
- Providing software suggestions and install required software on the team PCs and the virtual server.
- Contacting ITHelp on behalf of the team.
- Preparing and setting up document layouts and LaTeX documents on the team wiki and virtual server.
- Setting up and maintaining SVN and other software applications.

2.1.1.4 Lead Designer

The lead designer is responsible for producing a detailed plan of the system. This plan will be closely related to the requirements identified in the Software Requirements Specification document. UML design tools will mainly be used to assist in the design and presentation of the design. The lead designer will assign design tasks to the members of the design team and bring forth important design issues and ideas. She will have to work alongside the developers, since the design may have to be altered if it is found to be infeasible during the coding process.

2.1.1.5 Lead Coder

The lead coder will be responsible to update the team on the progress of the code during formal meetings. He will be working very closely with the lead designer to ensure that the lead designer's design is feasible throughout the entire design process. The lead coder will be doing the majority of code, with the help of another member - Lawrence.

2.1.1.6 Lead Tester

The software test leader is responsible for quality testing the software according to the Test Plan and QAP. This is done parallel as the software is being coded. He will not be part of the implementation team as to remove personal attachment to the code and not being bias.

2.1.1.7 Risk Manager

Monitors the risks associated with the activities and processes in use. Risk Manager also monitors and analyzes risky situations and assists the Project Manger to control the situation.

2.1.1.8 Minutes Taker

The main role of the Minute Taker is to take minutes for formal team, supervisor and client meetings. The Minute Taker will attend all formal meetings and take down correct information and decisions discussed in the meeting. He/She will be required to formally document the minutes and send a copy to all team members and other attendees (if any) via email within 48hrs.

2.1.1.9 Team Members

Team members are made up of separate individuals in a specific team. Activities of the team members entail:

- Executing activities assigned by the Project Manager.
- Writing short progress reports once week on the assigned.
- Reporting problems and concerns as early as possible.
- Working according to the guidelines listed in this document.

2.1.2 Weekly Timetable

The following timetable shows the common breaks team members share, as well as common breaks they share with the supervisor Nick. This allows for easy reference when a meeting is to be scheduled.

Time	Monday	Tuesday	Wednesday	Thursday	Friday
9am					
10am		Lawrence, Weng Hoe	Jinita, Weng Hoe, Jack	Jinita, Weng Hoe, Jack	Jack, Jinita, Weng Hoe
11am		Lawrence, Jinita, Jack	Lawrence, Jack	Jack	Lawrence
12pm	Lawrence, Jack, Weng Hoe	Lawrence	Lawrence, Jinita	Lawrence, Jinita, Weng Hoe, Jack	Lawrence
1pm		Lawrence	Lawrence, Jinita	Lawrence, Jinita, Jack, Weng Hoe	Lawrence, Jack, Weng Hoe
2pm	Jinita, Lawrence, Weng Hoe	Lawrence	Lawrence, Jinita, Weng Hoe		Lawrence
3pm	Lawrence, Weng Hoe	Lawrence	Lawrence, Weng Hoe	Jack, Jinita	Lawrence
4pm	Weng Hoe	Lawrence		Lawrence, Weng Hoe, Jack, Jinita	Lawrence
5pm					

Legend	
	Common breaks without Nick
	Common breaks with Nick
	Not a common break

2.2 Project Schedule

2.2.1 Gantt Chart

Please refer to `figs\Gantt.png` for the Gantt Chart. It is too long to be included on this page.

2.2.2 Task Table for Software Requirements Specification

1. Gather requirements through research, spec and client meetings. **(Done)**
2. Set up the layout on the wiki. **(Done)**
3. Define all the requirements in the SRS on the wiki. **(Done)**
4. Review the requirements for consistency, correctness and completeness. **(Done)**
5. Contact the client to clarify requirements that are inconsistent or incorrect. **(Done)**
6. Do an internal review by a team member who has worked least on the SRS. **(Done)**
7. Modify the SRS according to the suggested changes from Task 6. **(Done)**

8. Transfer the document from the wiki to LaTeX. **(Done)**
9. Send the SRS to the client and receive feedback. **(Done)**
10. Analyse and discuss the changes requested by the client. If the client requests to add more requirements, the team checks if it's feasible. **(Done)**
11. Modify the changes the team and the client agreed upon. **(Done)**
12. Send the SRS to the client to get it signed. **(Done)**

2.2.3 Task Table for Software Architectural Design

1. Each member puts forward a design architectural style that is best suitable for the project. Team holds a formal meeting to discuss the ideas and comes to a decision. **(Done)**
2. The team discusses and decides on design goals. **(Done)**
3. A sketch of the system is made with responsibilities, attributes and operations. **(Done)**
4. The sketch is verified with the requirements listed in the SRS and the coding team. **(Done)**
5. A domain model is drawn by the design team. **(Done)**
6. A class diagram is drawn by the design team.
7. Domain model and class diagrams are verified with the requirements listed in the SRS and the coding team.
8. Class diagram is validated to ensure that it meets the set design goals.
9. Class diagram is modified to ensure requirements and design goals are met.
10. Interaction diagrams are drawn to show the control flow between classes.
11. Interaction diagrams are verified with Use Cases and the requirements from the SRS.
12. Design team discusses the checklist of SDD and assigns tasks to each member to document the given sections. **(Done)**
13. Team members document the SDDs. **(Done)**
14. SDD is reviewed by two members that worked on it the least to ensure the checklist is satisfied and the requirements and design goals are met.
15. If required, modifications are done to the SDDs.
16. The SDD is reviewed by the member who contributes least or not at all to the SDD.
17. If required, suggested modifications are discussed and then applied.
18. The design is sent off to the 440 student to be reviewed.

2.2.4 Software Requirements Specification

The SRS has now been signed off by Jason. Jason made some changes to the document on the spot and signed the changes off as he went. These changes will be reflected in the LaTeX document. Some minor changes to the SRS should also be done. For instance some changes that were (mysteriously!) left out, like having a 'Browse' button instead of just a button. It shouldn't be a problem to get Jason to sign this off again some other time. It's no big deal and should not affect him at all.

2.2.5 Test Plan

The current Test Plan includes acceptance test cases and a module level test plan. The module level test plan is undergoing the review process, and will be revised once the review is done. The Test Plan will be changed accordingly as the design of the system grows.

2.2.6 Prototyping easyBLAST

Lawrence will be prototyping easyBLAST incrementally. This is also part of the testing phase to ensure that all our requirements can be met. Jack will assist Lawrence in prototyping.

2.2.7 Software Architectural Design

Jinita and Weng Hoe will work on the initial architectural design after. Jack and Lawrence will review design artifacts to ensure its feasible and consistent with the requirements in the SRS.

2.3 Technical Description of the Proposed System

One of the research activities conducted by the biologists of Peter MacCallum Cancer Centre is to regularly compare short fragments of sampled DNA sequences to an existing library of known DNA sequences. This allows biologists to study genetic activities that have occurred in the sampled individual. BLAST is used to carry out the sequence comparison process. Prior to applying BLAST, data is trimmed as desired to remove junk data and also to be manipulated so as to extract the regions of interest that have occurred in the sampled individual. One type of regions of interest is the unique identifier that was attached to the sequences during the experiment and the other type is the region comparable to the library of known DNA sequences. A piece of software has been requested to be built in order to simplify their job and to reduce operational costs.

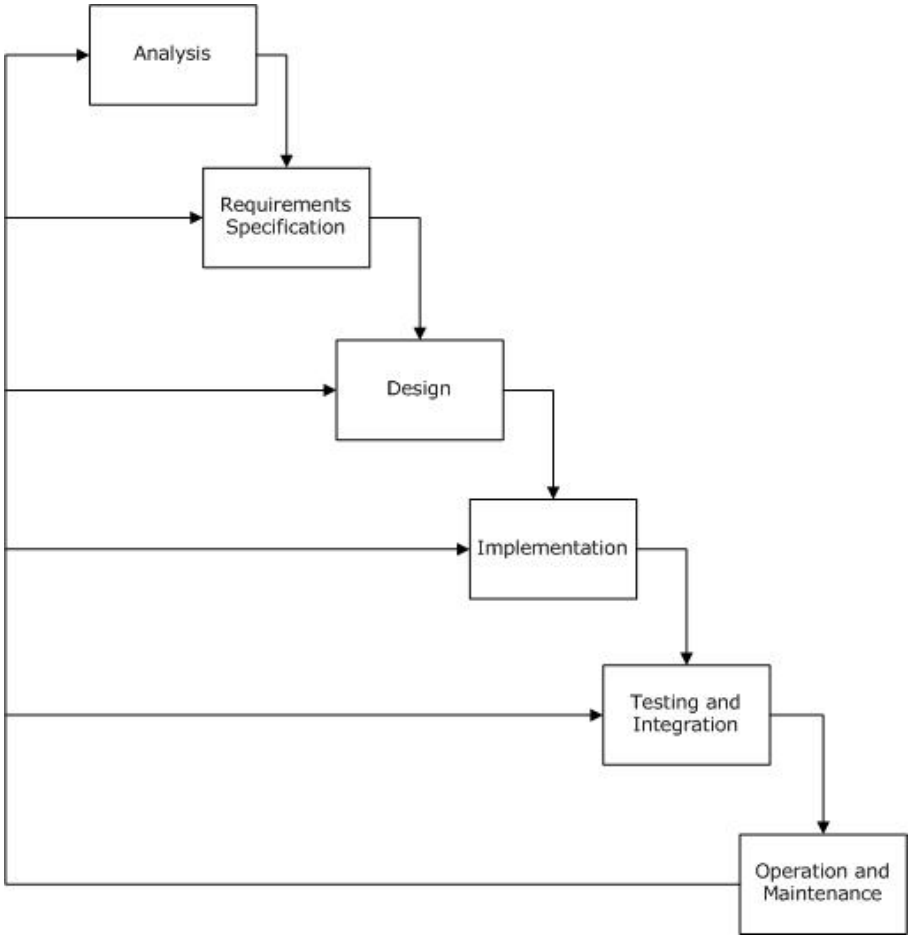
Moreover, the client is expecting more new data from external institutes to aid them in their research. This means that the amount of data they have to work with will eventually increase. This piece of software is meant to simplify their work in the future.

THREE

Project Standards, Procedures and Proposed Tools and Techniques

3.1 Process Model

The process model that will be used is a modification of the Waterfall model. It contains every stage the Waterfall model contains, which are Requirements, Design, Implementation, Testing and Maintenance. However, previous stages may be revisited at any stage of the process model. For example if a design problem is identified during implementation then the design can be modified. This model will be ideal for this type of project as there is a great deal of complexity with a lot of room for error. This process model enables errors in early stages of the model that may be discovered in the later stages of the process model to be corrected.



3.2 Communication Process

This section explains how meetings are conducted, how communication is established and how decisions are made.

3.2.1 Meetings

Since team members share a lot of common breaks together, meetings can be held fairly easily. Informal meetings are often held during these short common breaks. These informal meetings are used for simple discussions and also for the members to work together.

If necessary, formal meetings are held usually on Thursdays at 2pm. Formal meetings differ from informal meetings in the sense that it is where more important discussions are held and also important decisions are made. An agenda is created detailing the time and place of the meeting and items to discuss and resolve. A copy of the agenda is sent out to all attendees of the meeting via email at least 24hrs before the scheduled meeting time. Minutes are written during the meetings and are saved in the group's folder for future references and to provide records for members who are absent or unable to attend the meetings.

3.2.2 Communication

Instant messaging is sometimes used for discussion, but due to the ambiguity it can cause, it is generally not preferable. When a decision is made in an informal meeting (whether it be in-person or via the Internet), it is also sent to the team's email account to be documented. This way it can be easily referred to if necessary.

3.2.2.1 Emails

All team members must respond to every email sent by a fellow team member directed to them as a way of informing the sender that they have read the email. A simple "Read." is acceptable. This will make it easier for team members to discuss matters addressed over email in person, so they know whether a person has read it or not. All emails must be sent to the s340gd team email, and also every member's individual Gmail addresses, as there have been many times where the CSSE emails have not been forwarded to Gmail properly.

The following tags must be put in subject fields under certain circumstances:

- [340] - All emails related to 340 must include this tag.
- [SUP] - All emails directed to Nick Chadwick must include this tag.
- [TASK] - All emails concerning tasks (whether assigning tasks or responding to tasks assigned) must include this tag.

- [REVIEW] - All reviews must be sent out with this tag. Emails responding to reviews must also contain this tag.
- [JACK] - All emails specifically directed to Jack must include this tag.
- [LAW] - All emails specifically directed to Lawrence only must include this tag.
- [WH] - All emails specifically directed to Weng Hoe must include this tag.
- [JINI] - All emails specifically directed to Jinita must include this tag.

3.2.3 Decision Making

During meetings, each member has a chance to present their ideas. These ideas are expected to be challenged with a series of questions from other members. The presenter must then find ways to defend his ideas as a means of justifying them. This way, holes in ideas can be discovered early on, and members are given a chance to carefully think of their ideas before presenting them.

3.3 Tools

This sections lists what tools the team will use and its functionality throughout the year in order to produce the desired software. The risks and problems associated with the tool has also been explained.

- The Team Wiki will be used as a collaborative writing tool for new documents, where members responsible for the documentation will be able to collaborate effectively. Members should take note that conflicts can arise when more than one person is editing the same page at the same time. Once a document has been converted into LaTeX, it must be edited on LaTeX only. The outdated documents may still remain on the Wiki for historical purposes only. They are not to be worked on.
- LaTeX will be used to prepare pdf documents by transferring documents written on the Wiki, format them and convert them to pdf format. A LaTeX document layout will be setup in the SVN repository so that the members can work on different sections of the document without any conflict arises.
- The technical officer has setup an SVN repository in the team directory. The SVN repository will be used to keep track of the different versions of the documents. It will be used greatly when the implementation stage starts and the members begin to code.
- The Trac Project is a web-based management of the software project. It will be used to track tasks, monitors goals and to track bugs in the program. It provides an interface to subversion which allows the development team to reference tickets in the SVN log messages. It also has a ticketing system which acts as a bug tracker that will be used during the testing stage. Since Trac will be used by all members, it is important all members are familiar with the program and its features.

- Microsoft Visio and Microsoft Project will be used for generating diagrams. Since most of these diagrams will be done by one person only, it is not important that everyone is familiar with the same software, because there will not be much collaboration involved that needs a project file of the same extension.
- Gmail is the choice of email client for sending and receiving emails. Gmail's filter allows members to easily separate emails regarding the project from other emails. Gmail's search function is also very beneficial in finding a particular email. All mails must include the [340] tags in the subject field.
- The system will be coded in Java. This language is chosen for its ability cross-platform feature, most importantly. The project team is also familiar with the language since the members have previously done a subject that focuses on Java. The only IDE to be used is NetBeans 6.7. All members are expected to be familiar with NetBeans, so anyone can take over any of the current programmer's work very easily if necessary (for instance if the programmer falls sick).

3.4 Processes associated with team members

This section administrates a general guideline that every team member must follow. It also describes the processes that will be carried out if one fails to follow the protocol.

3.4.1 Attendance

Formal meetings held between team members must be attended by every single team member. If any member is unable to attend a scheduled formal meeting, the member must let the project manager know as soon as possible, so the meeting may be rescheduled. Due to the busy schedule of the client and the supervisor, it may not be possible to hold a meeting where every team member can be present, so meetings with the client or the supervisor will be scheduled based on the times the most team members can attend.

If a member does not attend a meeting without informing anyone beforehand, he will be sent a warning. After three of these warnings, the supervisor will be alerted to take appropriate action.

3.4.2 Sickness/Compassionate leave

Should the member become ill during the project and feel unable to meet the deadline, notify the project manager immediately and another member or more resources will be made available to help complete the task.

Should the member need to leave project for compassionate issues, the project manager-must be notified immediately. Member will need to brief replacement on progress as well as task at hand as well as provide an estimate on time required. This decision will be agreed upon by the project manager.

3.4.3 Heavy workload

All members must keep the project manager updated on the workload for all their subjects. This will enable more reasonable task scheduling to be done. If external workload is unexpectedly heavy, members should inform the project manager as soon as possible so work can be rescheduled and deadlines can be changed accordingly. If a member is unable to handle the workload of this project, the member must also inform the project manager. A negotiation can be carried out to try and meet a compromise.

Heavy workload is often due to poor person-job fit, so a fix may be for the project manager to schedule tasks based on person-job fit, i.e. if a person is better at carrying out a certain task, they should be allocated to that task instead of a task that they are not as good at. This will ensure that work is done in the most efficient and effective way.

3.4.4 Lateness

Members more than 5 minutes late to formal/informal meetings will be responsible for bringing a pack of lollies to the next formal meeting for the members that are on time (the lollies can still be eaten by the person that provided them if the other members that are on time agree to this). The type of lollies to bring will be determined at the end of the meeting. The decision will be made by the members that arrived on time. If the type of lollies to bring is accidentally undecided after the meeting has finished, the late member(s) will bring a pack of (at least) 55G Skittles by default.

3.4.5 Deadlines

All members are expected to adhere to the deadlines provided. If a time is not specified, the deadline will be at 5pm of the day specified. 5pm is the usual time most of the team members' subject submissions are due, so it is what they are used to. If a deadline is not met, the members involved in the task will be confronted by the others. If a reasonable explanation is not provided, the members will be sent a warning email. If a member has accumulated up to at least three unsatisfied deadlines, the team will have to alert the supervisor of such behaviour.

However, if a deadline is unreasonable and/or a team member decides that he needs more time for the particular task, the member will approach the project manager (if the member is the project manager, he will confront the rest of the team) and explain why the deadline is unreasonable. Once this request is approved, the project manager will reschedule the tasks at hand and change the deadline. A request to extend a deadline should at least be sent 48 hours before the deadline.

If a flexible deadline is provided (e.g. "Try and finish this by Thursday. Friday is fine too, but do your best to finish it by Thursday."), the team member will only have to send a simple email saying "I don't think I can finish this on time. Friday night is fine though.". These requests need not expect an approval response.

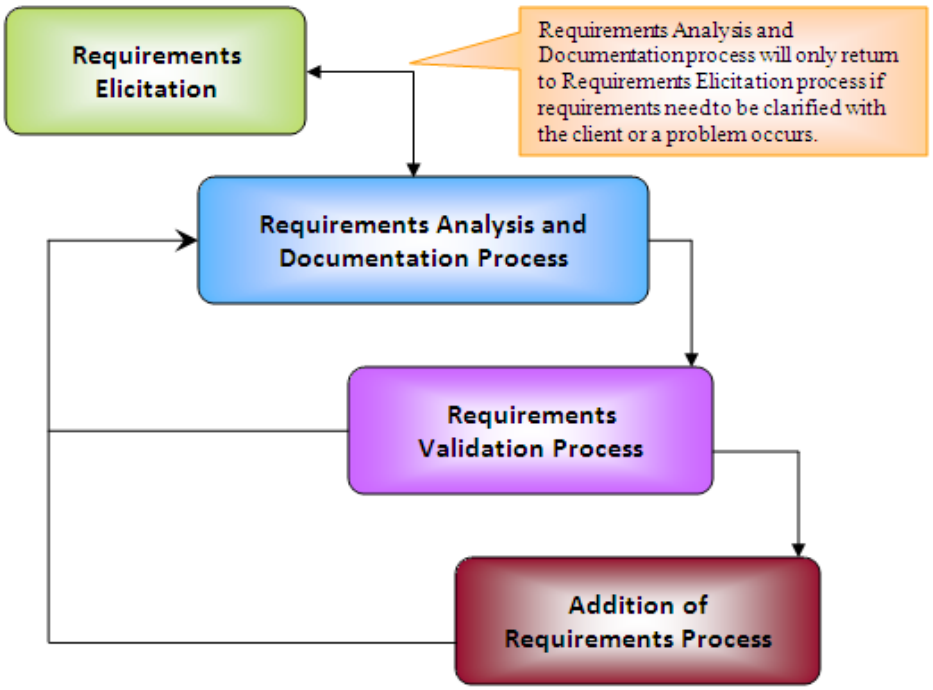
A punishment of giving the member that has not met a deadline more work is not going to be practised. This punishment will increase the workload of the member, and there

will be a high chance of the member not finishing any task at all due to the heavy workload. Therefore, sending three warnings before sending an alert to the supervisor is the most reasonable way of dealing with this.

FOUR

Requirements Plan

The requirements plan describes processes to be followed in gathering, analyzing and documenting the requirements efficiently and appropriately. The process of adding more requirements during later stages of the software cycle is also included to ensure team members follow the correct procedure to implement the new requirements. This is important as the process ensures the new version of requirements is consistent, correct and complete.



The figure above shows the Requirements Process the development team will follow. Refer to section 7.1 to read the Requirements Validation Process.

4.1 Requirements Elicitation Process

1. All team members will read the specs thoroughly and make a list of questions regarding the spec eg: sections they are unsure of.

2. Background reading on BLAST, DNA sequencing, FASTA and sHRNA will be done to be familiar with common terms and concepts. Reading of 341 lectures will also be done to assist the team with gathering and analyzing requirements.

3. A formal meeting will be scheduled to discuss the specs. Questions and Answers session will be held to share information as well as explain to other members of sections one is familiar with.

4. For requirements that all members are unsure of or do not fully comprehend will be noted and discussed in the client meeting. All members will be expected to attend the meeting.

5. A client meeting will be scheduled to confirm the team's understanding of the problem and the list of requirements with the client. A questions and answer session will be held to gain better understanding of the current system used at Peter MacCallum as well as to gather more information on user's wants and expectations.

6. If any confusion or loop holes regarding the requirements is identified, the client will be contacted as early as possible. The problems is addressed either by exchange of emails or through a formal client meeting.

4.2 Requirements Analysis and Documenting Process

1. Team members of the Requirements Team will construct the layout of Software Requirements Specifictation (SRS) document. They will then distribute sections of SRS and the requirements that each individual will work on.

2. During the process of documenting the requirements, if any requirement are incorrect, inconsistent or the member does not fully comprehend the importance or the function of the requirement in context of the problem, other team members are approached. If required a formal meeting is called to analyze the problem and possibly propose a solution.

3. If no confirmed solution is found by the team, the client is contacted via email.

4. Requirements team will handle any problems associated with the SRS. This may often require repeating steps 4, 5 and 6 of the Requirements Elicitation Process.

4.3 Addition of Requirements Process

This process describes the steps that will be taken when requirements are added or modified to the current SRS, during the later stages of the software life cycle. The procedure only handles the Requirement stage. The process of implementing into other completed stages will be handled by the individual stages. For example, after the modification to the requirements have been confirmed, Modification of Design process will in turn take over to implement the new requirements into the current design. This process is to ensure the new requirements of the system are still consistent, correct and complete.

FIVE

Design Plan

This section describes proccesses that will be used by the design team to design the system. The design team will follow the deisgn process described below to create a complete and feasible design of the system. Domain models, interaction diagrams and class diagrams will be constructed to represent the design. Object-oriented design methodolody will be used. Modifying Design process will be followed when requirements are modified to implement the changes in the design.

5.1 Design Process

1. The development team will first discuss the design goals of the system and agree upon a particular architectural style. After that, a sketch of the system will be drawn including the responsibilities and operations.

2. Using the requirements list in the SRS and the sketch, the design team will construct a domain model of the system. Domain model will provide a higher-level view of the system and assist the team to identify objects, classes and links between them.

3. A internal review will be done by the team member who is not in the design team. The feedback will be discussed and agreed changes will be implemented. This review process is important as it ensures the design team has considered all the requirements, major elements have not been missed and the design is feasible.

4. Using the domain model, a detailed class diagram of the system will be drawn. This will ensure that an easy transition is made from the design to the coding stage and ensure each class has suitable functions to carry out its responsibility.

5. The class diagram is verified and validated by the coding team to ensure its feasible and all requirements are met.

6. From the use cases describes in the SRS, interaction diagrams are constructed for each scenario described by the use cases. If any scenarios that were overlooked, will also be included.

7. Interaction diagrams are reviewed by the design team to ensure that the program execution flow is feasible and the diagrams are consistent with the class diagram.

8. An internal review will be done to ensure all the requirements have been satisfied and to ensure that the design is feasible.

9. The design team will document the design and ensure the checklist is satisfied. An internal review will be done and the suggested modifications will be carried out.

5.2 Modifying Design Process

The process is followed when requirements are modified during the later stages of the software cycle. For example extra requirements are added due to the client's request during the implementation stage of the project. This will ensure the appropriate modifications are made to the design.

SIX

Implementation Plan

6.1 Implementation Process

The implementation of the design will follow the design very strictly. Infeasible parts of the design discovered during the implementation process may be changed. Once a new version of the design is complete, the implementation can resume work. The lead coder will be responsible for leading the implementation stage. He will be in charge of overlooking the implementation stage, and allocating tasks related to the code to the rest of the team.

Documentation must be done during the coding process, not after. Since a lot of collaboration between team members will be involved during the coding period, documentation is crucial for everyone to have a great understanding of it. Not all members will be coding together, since other work will be done in parallel. Members that have completed other tasks not related to the code will be reviewing the code as the coders are working on the software.

JavaDoc will be written alongside the code.

The Java Naming Convention should be used for naming variables, objects, etc. and also when referring to these things in the logs. Please refer to the following link:

`http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html`

For instance, do not name variables as 'f' instead of 'file' and 'in' instead of 'inputFile'.

No lines should exceed the 80 character limit.

SEVEN

Quality Assurance Plan Chapter

All reviews must be sent by email. If the review documents are too large for the CSSE server, they will be sent to all members' Gmail accounts. In addition to that, all reviews are to be put in `\team\review`. The naming convention used for the files will be: `340_[Artifact]_Review-[CSSE login].pdf`. Any subsequent reviews done to the same document will be named `340_[Artifact]_Review-[CSSE login]_[number].pdf`. For instance, if Weng Hoe has reviewed the SDD for the third time, the file will be named `340_SDD_Review-wengn_3.pdf`. If Jack has reviewed the SRS for the first time, the file will be named `340_SRS_Review-wjlow.pdf`.

Each item in the review will be labelled:

- MINOR: Miniscule detail that should be altered to increase the professionalism of the document.
- MAJOR: Important detail that must be altered because it challenges the author's understanding of the system.
- MAYBE: A problem that the reviewer is unsure of. The person in charge of implementing the changes must decide upon whether this is an issue or not. The entire team may be brought in to decide upon this if necessary.

Please refer to `340_TP_Review-lbang.pdf` for a template to use. Font style and font size conventions are not in place. Use anything that's readable. There is also no strict text formatting necessary.

7.1 Design Validation Process

1. An internal review will be carried out by the two members that have worked least on the design.
2. The design leader will implement the necessary changes based on the reviews.
3. If the design leader disagrees with any part of the reviews, the design leader will respond to the review by email.
4. If the reviewer accepts the design leader's explanation, the change will not have to be made. Otherwise, the rest of the team will be consulted to settle on the dispute.

5. Once the changes have been implemented, a quick informal review will be done by one of the two reviewers.

6. Any necessary changes will be discussed with the design leader and implemented accordingly.

7.2 Code Review Process

Members not directly involved in the coding process will be in charge of reviewing the code. The reviewer will compile a list of questions regarding the code that are not answered in the documentation, e.g. "Why is this here?", "What is this block of code for?", etc., and expect the members involved in the coding process to be able to answer these without ambiguity. Questions that are unable to be answered should be taken note of. For instance, if the programmers cannot answer the question as to why a particular class exists, maybe the class is redundant.

The reviewer will also give feedback on structure and the documentation of the code. Structure of the code involves ensuring that indentations, spaces and a consistent format has been used. This is to ensure that the code is readable for the testing and the maintenance team. Documentation of the code involves ensuring that comments and explanation of functions and objects are suitable and correct. As well as, appropriate headers are included in each file with a summary of the objective of the file and variables and functions are given suitable names. This is again to ensure that the code is comprehensive to the testing and the maintenance team

7.3 Testing Process

The V-model will be used as a guideline for the entire testing process. Once the requirements have been analysed, acceptance test cases will be written. Subsequently, the system design will take place, followed by the writing of the system design test cases. After that, the architectural design and the module designs will be completed. The integration tests and the unit tests will be written out after that. The coding process follows closely after.

After the code has been written, unit testing, integration testing, system testing and acceptance testing will be carried out in sequence. These tests will be done according to how they have been written before the coding process started.

When the coders has finished writing a function, they are required to send a ticket using Trac to the test leader for testing. The test leader can use the same ticket to report the testing results and fix the faults if any. The following template will be used to assign a testing ticket. The test leader is required to accept the ticket first before proceeding to testing.

Short summary: Testing - [Method name] in [Class name]

Component: Testing

Type: task

Full description:

- Explain how the function works

Priority: major

Severity: major

7.4 Bug Reporting Process

When bugs are encountered by the software test leader, bug reports are sent to the lead coder. How the bugs are reproducible should be written specific and straight to the point. To make sure the bug really exists in the software, the test leader should reproduce it at least three times before lodging the report. Using Trac, the test leader shall send a ticket to the coders to explain what is wrong with the code. If the coder has opened a ticket for testing, then the same ticket will be used to report. For keeping records of the report, the following template will be used to report bugs. If there are files generated from testing, it will also be attach to the ticket to help the coders to understand the problem. The ticket opened can only be closed by the test leader after the defect is fixed and tested.

Short summary: Defect - [Method name] in [Class name]

Component:

Type: defect

Full description:

- Explain what is wrong
- Output generated by the testing tool

Priority: major

Severity: major

EIGHT

Configuration Management Plan

8.1 Version Control

Subversion will be used as team's version control software. Documents that will require version control are all deliverables and code for the actual software.

Every version of every deliverable must be put into the team folder. A "version" in this case refers to a version that has major changes, for example an SPMP that must be re-read by every member of the team. Several typo fixes do not constitute a "version". Please refer to the following email sent by Lawrence for processes related to this:

—

The email folder is now password-protected, and you may log in with the usual username\password combination.

*There is also a deliverables folder which is password-protected. The deliverables folder should contain each artifact by version and the *latest *copy as well. *In the future, PRIOR to any version change to a deliverable, it should be compiled and placed in the team/deliverables folder with appropriate permissions.* The latest copy is also placed here, and if an artifact exists with the same version, but lower revision number, then it should be deleted. The version number should be updated when there is a significant change (your discretion) to the document.*

Deliverables are currently named as: ARTIFACT_vX.Y_rREV_YYYYMMDD_DinoPirates.pdf

This is so everyone can see the latest change made to a document and assessors will know the authors.

—

The original email can be read here: `http:\\s340gd.cs.mu.oz.au\email\08\0024.html`

If a problem should arise with the version control software, that user must:

1. Contact the technical officer of the group.
2. Inform the technical officer of the problems and its details.
3. The technical officer will then seek to remedy the solution as soon as possible.

Should the version control software become beyond reparable, the team must

1. Figure out who has checked out the latest revision.
2. Have the technical officer install a fresh copy of the version control software
3. Import the latest local copies on to the repository.

8.2 Backups

Backups will be done daily and stored on the team folder. Daily archives of the past 14 days will be kept, and after 14 days, a weekly archive will be kept instead of daily. Team members should copy these archives regularly onto their flash drives, preferably at the start of the week(Monday) and at the end of the week(Thursday or Friday). That way, we have more than one copy that is not dependent on the server being available. Once a backup to flash drives is done, the team member is required to sent out an email to inform everyone of the backup. The mail must contain the tag [BACK] and a link to where the backup is hosted(only for backup at the end of the week).

Full backups are available from: http://s340gd.cs.mu.oz.au/backups/

Should the server become unavailable "'permanently"', the following steps should be taken:

1. Figure out who has a copy of the latest archive.
2. Restore the previous archive

If the server is unavailable temporarily, then the severity of the outage must be determined. If it's a short while, the team may take a break; if the outage is spans over a few days, the team should decide whether to work off the old revision, or apply the changes to the new one.

8.3 Mail

Mail sent to and from the group are tracked and archived into HTML via hypermail. As minutes and agendas are sent often, members will have access to the mail archive, allowing to see what has been agreed upon. Mail sent must contain the tag [340] in the subject field to distinguish itself from the amount of same received via the same address.

If mail is sent without the correct [340] tag, they should be resent so it can be properly archived.

If for whatever reason, a member does not receive an email directed at all members, they should check the archived HTML copies. If everyone has received an email, but it has not been archived, then it should be sent again to the person in charge of the archiving.

Actual emailing processes are documented in section 3.2.2.1.

8.4 IDE

NetBeans will be used with default settings as our IDE once we begin developing code. Having the same settings allows for everyone to be on the same page should someone get stuck. If a default setting is deemed counter-productive by the team, the setting will be changed.

Should NetBeans become inoperable to work on for whatever reason, the team must decide on a new IDE to develop the code in. A possible choice is Eclipse.

NINE

Documentation Plan

9.1 Document Writing Process

The ConOps, SRS and Test Plan were done on the wiki before being transferred to LaTeX. However, this practice proved to be troublesome. If multiple people were working on the same document, conflicts were hard to identify and resolve. Since every member has since learned to work with LaTeX (instead of only Lawrence and Weng Hoe before this), the team has now switched to working straight in LaTeX using SVN repository. The nature of SVN means that conflicts can be identified and resolved easily. Older versions of files are also easier to retrieve this way.

Everything written is expected to be as typo-free as possible, especially for documents that are being collaborated. It is very difficult for collaborators to add to the document if parts of the document are unreadable or hard to understand. Run a spell-check if possible!

9.2 Document Review Proces

Since the team is usually split up to work on multiple documents in parallel, once a document is written, it should undergo an internal review process. This internal review will be done by the people that did not work on the document directly. This is to reduce biases that may arise when an individual reviews his own work. Once an internal review has been done and the document is approved, it will then be submitted for an external review. It will be sent to different people depending on when the team expects the external review to be completed. Having an internal review before submitting a document for an external review increases the sense of professionalism that the external party will perceive in the project team.

9.3 Code Documentation Process

Documentation of the code will be written in JavaDoc for each class and function. A short description of each class should be prefixed to every *.java file and each funtion is to have it's description, parameters, outputs and exceptions detailed. Comments (not JavaDoc) should also be made where the code may be ambiguous or seem out of flow. This should be added at the author's discretion.

TEN

Resource Management Plan

10.1 Work Division

If work can be done in parallel, work can be divided among team members. Each team member's preference is emailed to the project manager (Jack). The project manager then tries to divide work up based on the team members' preferences. Some members' preferences may have to be compromised in certain cases.

10.2 Data Files Backup

A shell script is run at 8pm everyday to create a backup of the team folder. Another way of backing up is to upload the files onto Gmail. This is quite safe as Google has always been very reliable. Even though this is safe, it is not exactly well-organised. Hence, the files uploaded to Gmail are also stored on each team member's home computers. The backup directory contains a list of months of the year as directories. In each month directory, there exists directories that represent the dates the files are backed up.

ELEVEN

Risk Management Plan

11.1 Slacking member

Description: A team member not doing his part of the work or not working as much as expected/necessary.

Probability: Medium

Impact: Medium: Time slippage, low quality product.

Mitigation: Have every member understand what other members are doing so they can take over others' work easily.

Containment: Reallocate workforce. Have another member take over or assist the slacking member in finishing the work.

11.2 External workload

Description: Assignments and projects from other subjects turn out to be too overwhelming alongside this project.

Probability: High

Impact: Medium: Time slippage.

Mitigation: Members that are doing common subjects should help each other out. Every member should also understand what other members are doing so they can take over their work easily. If any member has less work than the others for a particular period, he should assist the others in their work for this project so as to relieve their pressure.

Containment: Reallocate workforce. Have another member take over or assist the busy member in finishing the work.

11.3 Interaction with BLAST

Description: Having the program interact with BLAST without any modifications to the BLAST program appearing to be too difficult.

Probability: Low

Impact: Critical: Unable to meet basic system requirements. Final product not meeting the specifications.

Mitigation: Do more research on BLAST and have the client demonstrate the running of the BLAST program.

Containment: Change the specifications and system requirements.

11.4 Data loss

Description: Data loss from failure of the hard disk where data is stored.

Probability: Medium

Impact: High: Time slippage, having to redo a lot of work.

Mitigation: Daily scheduled backups should be done. Weekly backups should be stored in several different physical locations.

Containment: None.

11.5 Simplicity issue

Description: End users being unable to handle the new system.

Probability: Low

Impact: High: The system, which is to simplify work, turns out to be doing the opposite.

Mitigation: During the design of the new system, the potential users should be surveyed to provide feedback on the design to ensure that they are happy with adapting to / using it in the future.

Containment: The project team provides training to the users.

11.6 Internal workload

Description: Workload is underestimated for a particular stage of the project.

Probability: Medium

Impact: Medium: Time slippage. Tasks having to be reallocated and possibly redo the schedule for that particular stage. More work is assigned to team members.

Mitigation: For each stage, processes are monitored closely and all tasks are listed before a stage commences. Information is gathered on the tasks, its importance and estimated time required to complete the task is evaluated. If the task is critical and is time consuming, several team members will be assigned to it.

Containment: Workload of the stage is reanalyzed. If required, workload is redistributed amongst the team members. Schedule is modified and process for schedule slippage is carried out.

11.7 Schedule slippage

Description: Schedule slippage occurs. This is when a task or a particular stage is behind schedule.

Probability: Medium

Impact: High: Dependent tasks and stages are affected. Team members may have insufficient time to complete dependent tasks and stages.

Mitigation: Processes for tasks are followed and monitored. The schedule will include the worst dates of completion of tasks but the team will aim to complete the task before the given date.

Containment: The progression of the task is analyzed by project manager. If the task is critical and the progress of it is unsatisfactory, project manager will warn the team member and monitor the him/her closely. If required, an extra team member will be assigned to help complete the task.

11.8 Missing member

Description: A team member is uncontactable during a critical period.

Probability: Low

Impact: High: The schedule for the project, tasks and stages the member was in charge of will be affected. Schedule will have to be redone and tasks and stages will have to be divided amongst the team members. This will increase the workload of team members.

Mitigation: Project Manager has emergency contact numbers of all team members. In the case of Project Manger being uncontactable, a team member will have his/her emergency contact number as well.

Containment: The supervisor will be informed of the situation. All the information on the progress of the tasks the member was working on is gathered by the Project Manager and allocated to other team member(s). Tasks and stages that the missing team member was the in charge of will be reassigned.

11.9 Processes not followed

Description: A team member is not following the processes properly.

Probability: High

Impact: High: Dependent tasks that are affected by these processes being severely affected, causing time slippage, etc.

Mitigation: Monthly audits should be done to keep team members on the right track before things go horribly wrong.

Containment: Rescheduling of tasks and workforce.

11.10 Additional requirements / Change in requirements

Description: More requirements are added or changed during the later stages of the development life cycle.

Probability: High

Impact: High: Time slippage, possibility that other requirements have to be changed to meet new requirements.

Mitigation: Inform the client that future requirements will have to be optional requirements that are only done if time permits. Keep the client updated regarding what requirements can or cannot be added/removed during a current stage. Design a system with low coupling and high cohesion.

Containment: The design of the system must be revised to ensure that the new requirements and the design are consistent with each other.

11.11 Design not meeting requirements

Description: Design does not meet the requirements stated in the SRS.

Probability: Medium

Impact: Critical: The end system will not function according to the clients requirements.

Mitigation: During each stage of the design, SRS will be referred to and continuous validation checks will be done.

Containment: Design will be reanalyzed to evaluate the effect of fixing the design to meet the excluded requirement. Changes are made accordingly and validated using the validation and review processes.

11.12 Unable to implement all essential requirements

Description: The team is unable to implement all the essential requirements in the design.

Probability: Medium

Impact: High: Development team will be compelled to repeat the requirement's and design processes to fix the problem. This will cause schedule slippage if identified in later stages of the software cycle.

Mitigation: In requirements validation process, several reviews will be done including a feedback from the supervisor and the client to ensure the requirements are consistent, correct and complete. During each stage of the design, continuous validation checks will be done. Requirements will be written, keeping in mind the design and implementation stages.

Containment: Explanation and suggestions regarding the requirements will be put forward to the client. The team will suggest modifications that would be best to make to construct a feasible design. The suggestions will also be such that the end system will have the same functionality to what the client desired.

11.13 Overlooked test cases

Description: The testing team unintentionally overlooks important tests during the testing stage. This includes cases that will cause the program to crash and loose the data.

Probability: Medium

Impact: Critical: The end product may contain a large amount of bugs and errors.

Mitigation: The testing team will carry out intensive testing which includes having biologists test the system as well.

Containment: All test cases in the test plan will be run in sequence to ensure that every single one is covered.

A

Glossary

The following definitions, acronyms and abbreviations are used throughout this document:

1. **CSSE:** Department of Computer Science and Software Engineering
2. **GUI:** Graphical User Interface
3. **BLAST:** Basic Local Alignment Search Tool, or BLAST is a commonly used family of programs for matching DNA sequences.